实验目的

生成和显示分段三次B′ezier曲线、B样条曲线、旋转曲面（围绕y轴的xy平面上的曲线）和广义圆柱面。

曲线绘制

为表示曲线和局部坐标系，需计算一个点坐标V和三个向量坐标T、N、B。

Bézier曲线的表示式如下。

$$\gamma(t) = \mathbf{GMT}(t) \tag{22.6}$$

后面将介绍的所有曲线都具有如式(22.6)的构造形式，即包含一个几何矩阵 $\boldsymbol{G}$(通常包括四个点而不是两个点和两个向量)、一个基矩阵 $\boldsymbol{M}$(列出各相关多项式的系数)和向量 $\boldsymbol{T}(t)$，不同类型曲线间的区别在于几何矩阵所含内容不同、基矩阵对应的多项式不同。

22.3.1 Bézier 曲线

本小节介绍第二种曲线类型：Bézier 曲线。它由四个点 P_1，…，P_4 确定，曲线起点为 P_1，终点为 P_4，曲线在起点处的方向向量为 $3(P_2-P_1)$，终点处的方向向量为 $3(P_4-P_3)$，如图 22-3 所示。

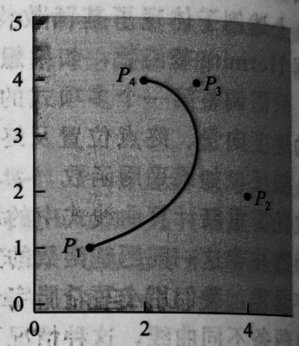


图 22-3 一条 Bézier 曲线图例，起点为 P_1，该点处切向量沿 P_1P_2 方向，终点为 P_4，该点处切向量沿 P_3P_4 方向

Bézier 曲线表示为

$$\gamma(t) = [P_1;P_2;P_3;P_4]\begin{bmatrix}1 & -3 & 3 & -1\\ 0 & 3 & -6 & 3\\ 0 & 0 & 3 & -3\\ 0 & 0 & 0 & 1\end{bmatrix}\boldsymbol{T}(t) \tag{22.7}$$

式中几何矩阵包含的是四个点，基矩阵中的系数也与 Hermite 曲线不一样。

求V对t的导数，可以得到法向量T。

$$P(t)' = [P1, P2, P3, P4]\cdot\begin{bmatrix}1 & -3 & 3 & -1\\ 0 & 3 & -6 & 3\\ 0 & 0 & 3 & -3\\ 0 & 0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}0\\ 1\\ 2t\\ 3t^2\end{bmatrix}$$

N和B不能简单地通过求导数计算得到；T′不一定垂直于T，乃至不一定存在。可以任选一个次法线方向 B_0=(0,0,1)，通过下式迭代求出各点处的N和T：

$$N_i = norm(B_{i-1}\times T_i)$$
$$B_i = norm(T_i\times N_i)$$

式中norm表示求单位向量。

据此可以得到如下代码。

```
Curve evalBezier(const vector<Vector3f> &P, unsigned steps)
{
    if (P.size() < 4 || P.size() % 3 != 1)
    {
        cerr << "evalBezier must be called with 3n+1 control points." << endl;
        exit(0);
    }

    Curve R;
    Vector3f B(0, 0, 1);

    for (size_t i = 0, numCurves = (P.size() - 1) / 3; i < numCurves; ++i)
        for (unsigned j = 0; j <= steps; ++j)
        {
            float t = float(j) / steps, t2 = t * t, t3 = t2 * t;

            Vector3f V = (1 - 3 * t + 3 * t2 - t3) * P[i * 3] +
                         (3 * t - 6 * t2 + 3 * t3) * P[i * 3 + 1] +
                         (3 * t2 - 3 * t3) * P[i * 3 + 2] +
                         t3 * P[i * 3 + 3],

                     T = ((-3 + 6 * t - 3 * t2) * P[i * 3] +
                          (3 - 12 * t + 9 * t2) * P[i * 3 + 1] +
                          (6 * t - 9 * t2) * P[i * 3 + 2] +
                          3 * t2 * P[i * 3 + 3])
                             .normalized(),

                     N = Vector3f::cross(B, T).normalized();

            B = Vector3f::cross(T, N);

            R.push_back({V, T, N, B});
        }

    return fixClosed(R);
}
```

其中fixClosed用来解决闭合曲线首尾法向量不一致的问题。记首尾法向量夹角为θ，曲线一共有n个（包含首尾）采样点，只需令第i个点（从0起）旋转

$$\frac{\theta \cdot i}{n-1}$$

```
Curve &fixClosed(Curve &c)
{
    // If curve is not closed or need not fixing, return it as is.
    if (!approx(c.front().V, c.back().V) || !approx(c.front().T, c.back().T) ||
        approx(c.front().N, c.back().N))
        return c;

    float t = acos(Vector3f::dot(c.front().N, c.back().N)) / (c.size() - 1);
    for (size_t i = 0; i < c.size(); i++)
    {
        c[i].N = cos(t * i) * c[i].N - sin(t * i) * c[i].B;
        c[i].B = Vector3f::cross(c[i].T, c[i].N);
    }

    return c;
}
```

由下面的公式，仿照上面做法，三次B样条曲线也可以轻松完成。

22.5　三次 B 样条

三次 B 样条(还有线性、二次、四次 B 样条等，但三次 B 样条是最常用的)与 Catmull-Rom 样条类似，但二者又存在两个重要的差异：三次 B 样条是 C^2 连续的，即它的一阶和二阶导数都是连续函数；三次 B 样条不是插值型样条，它通常靠近但不通过控制点。

三次 B 样条有两种形式，即均匀和非均匀，我们首先介绍均匀 B 样条。控制顶点为 P_0，…，P_n 的三次 B 样条的表达式为

$$\gamma(t)=\sum_{i=0}^{n}P_i b_3(t-i) \tag{22.17}$$

其中

$$b_3(t)=\begin{cases}\frac{1}{6}t^3 & 0\leqslant t\leqslant 1\\ \frac{1}{6}(-3(t-1)^3+3(t-1)^2+3(t-1)+1) & 1\leqslant t\leqslant 2\\ \frac{1}{6}(3(t-2)^3-6(t-2)^2+4) & 2\leqslant t\leqslant 3\\ \frac{1}{6}(-(t-3)^3+3(t-3)^2-3(t-3)+1) & 3\leqslant t\leqslant 4\\ 0 & \text{其他}\end{cases} \tag{22.18}$$

曲线 γ 的定义域为 $0\leqslant t\leqslant n-2$。由于函数 b_3 的构造方式，当 $j\leqslant t\leqslant j+1$ 时，$\gamma(t)$落在由四个控制点 P_j，…，P_{j+3}形成的凸包内，即**凸包性**(convex hull property)。这一性质在光线与 B 样条曲线求交时十分有用，若光线与由四个连续控制点形成的凸包不相交，则与该四个控制点确定的 B 样条曲线段也不会相交。因此只有当光线与凸包有交时，才需进一步进行与曲线的求交计算，具体细节可参考相关的网络资源。

与 Bézier 曲线和 Hermite 曲线一样，B 样条曲线段也可以表示成矩阵形式，从而提高计算效率。回顾 Bézier 和 Hermite 曲线的矩阵表示为

$$\gamma(t)=\mathbf{GMT}(t) \tag{22.19}$$

其中 $\boldsymbol{T}(t)$为 t 的幂向量$[1\quad t\quad t^2\quad t^3]^{\mathrm{T}}$。由于一条 B 样条曲线由多个曲线段组成，它们分别定义在 $0\leqslant t\leqslant 1$、$1\leqslant t\leqslant 2$ 等各节点区间内，我们以 $\boldsymbol{T}(t-j)$替换式(22.19)中的 $\boldsymbol{T}(t)$，即取参数 t 的小数部分来定义第 j 个曲线段。

第 j 段曲线定义在$j\leqslant t\leqslant j+1$ 区间，由控制点 P_j，…，P_{j+3}确定，其几何矩阵为

$$\boldsymbol{G}_B=[P_j;P_{j-1};P_{j-2};P_{j-3}] \tag{22.20}$$

对平面曲线而言，$\boldsymbol{G}_B$ 为 2×4 矩阵；对空间曲线而言，$\boldsymbol{G}_B$ 为 3×4 矩阵，各列为相应控制点的坐标。将 $\boldsymbol{G}_B$ 与 **B 样条基矩阵**(B-spline basis matrix)$\boldsymbol{M}_{\mathrm{Bs}}$：

$$\frac{1}{6}\begin{bmatrix}0 & 0 & 0 & 1\\ 1 & 3 & 3 & -3\\ 4 & 0 & -6 & 3\\ 1 & -3 & 3 & -1\end{bmatrix} \tag{22.21}$$

相乘，则均匀 B 样条曲线表示为

$$\gamma(t)=\boldsymbol{G}_B\boldsymbol{M}_{\mathrm{BS}}\boldsymbol{T}(t-j) \tag{22.22}$$

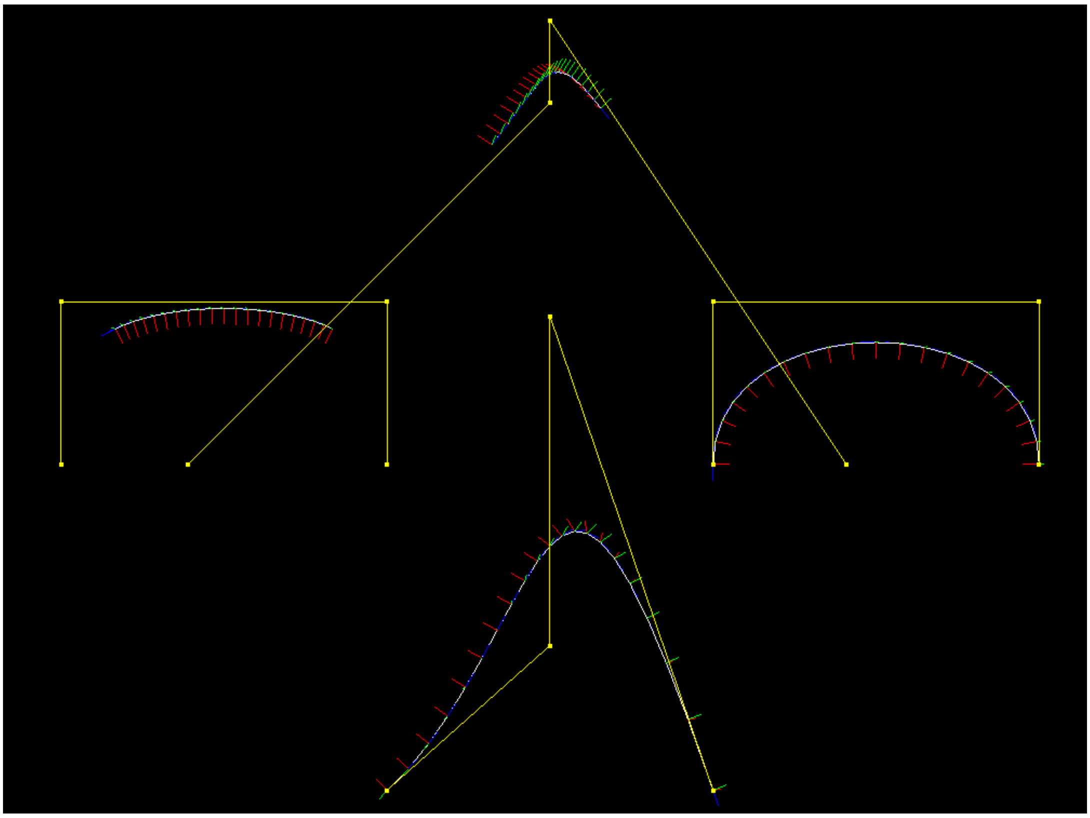
这里 $j=\lfloor t\rfloor$，因而 $t-j$ 是 t 的小数部分。

虽然 B 样条曲线不能插值各控制点，但它具有较高阶的连续性，这使得其为许多应用所青睐。如何权衡曲线形状的可控性(是否插值其控制点)和曲线的连续性(光滑程度)，是一个必须根据具体的应用情况加以考虑的问题。

代码与Bézier曲线大同小异。

```
Curve evalBspline(const vector<Vector3f> &P, unsigned steps)
{
    if (P.size() < 4)
    {
        cerr << "evalBspline must be called with 4 or more control points." << endl;
        exit(0);
    }

    Curve R;
    Vector3f B(0, 0, 1);

    for (size_t i = 0, numCurves = P.size() - 3; i < numCurves; ++i)
        for (unsigned j = 0; j <= steps; ++j)
        {
            float t = float(j) / steps, t2 = t * t, t3 = t2 * t;

            Vector3f V = (1.0f / 6.0f) * ((-t3 + 3 * t2 - 3 * t + 1) * P[i] +
                                          (3 * t3 - 6 * t2 + 4) * P[i + 1] +
                                          (-3 * t3 + 3 * t2 + 3 * t + 1) * P[i + 2] +
                                          t3 * P[i + 3]),

                     T = ((-t2 + 2 * t - 1) * P[i] +
                          (3 * t2 - 4 * t) * P[i + 1] +
                          (-3 * t2 + 2 * t + 1) * P[i + 2] +
                          t2 * P[i + 3])
                             .normalized(),

                     N = Vector3f::cross(B, T).normalized();

            B = Vector3f::cross(T, N);

            R.push_back({V, T, N, B});
        }

    return fixClosed(R);
}
```

用上面的代码绘制core.swp，效果如图。



曲面绘制

旋转曲面的生成方式比较直观，即将曲线上每个点绕y轴旋转，并保持法向量与切向量垂直。绕y轴旋转变换的矩阵为$M =$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

通常，法向量的变换矩阵为逆转置，即$(M^T)^{-1}$，推导过程附后。

特别地，旋转变换的矩阵是行列式为1的正交矩阵，即特殊正交矩阵，有$M = (M^T)^{-1}$。故法向量可以直接由$M \times n$给出，无需使用$(M^T)^{-1} \times n$。

据此写出旋转曲面的代码

```
Surface makeSurfRev(const Curve &profile, unsigned steps)
{
    if (!checkFlat(profile))
        throw "makeSurfRev: profile curve must be flat on xy plane.";

    constexpr float PI2 = 2 * 3.14159265358979323846f;
    Surface surface;
    for (unsigned j = 0; j <= steps; j++)
    {
        Matrix3f M = Matrix3f::rotateY(PI2 * j / steps);
        for (const auto &p : profile)
            surface.VV.push_back(M * p.V),
            surface.VN.push_back(M /*.inverse().transposed() */ * -p.N);
    }
    addTriangles(surface.VF, steps + 1, profile.size());
    return surface;
}
```

addTriangle用于定义朝外的三角形面元，应按照从外侧看逆时针的顺序给出各顶点。

```
void addTriangles(vector<Tup3u> &VF, size_t sweepSize, size_t curveSize)
{
    for (unsigned i = 0; i < (sweepSize - 1) * curveSize; i++)
    {
        if ((i + 1) % curveSize == 0)
            continue;
        VF.push_back({i, i + 1, i + curveSize});
        VF.push_back({i + 1, i + curveSize + 1, i + curveSize});
    }
}
```

广义圆柱体

与上类似，只是把旋转变换矩阵R_y替换为扫描曲线的局部坐标系矩阵

$$M = \begin{bmatrix} N & B & T & V \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

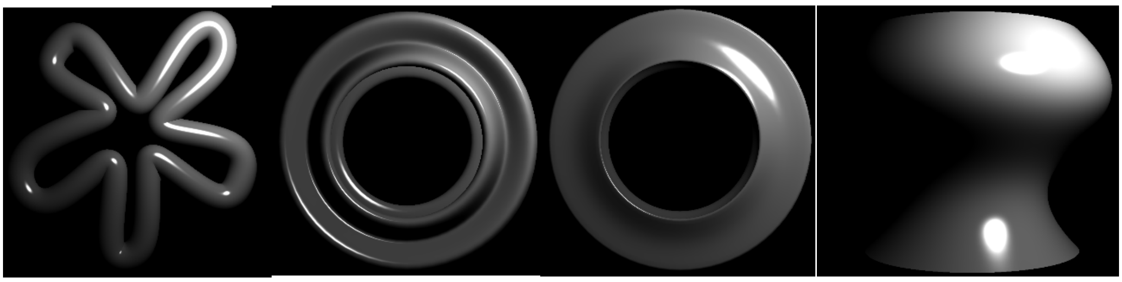
按照一般仿射变换的公式，写出这样的代码

```
Surface makeGenCyl0(const Curve &profile, const Curve &sweep)
{
    if (!checkFlat(profile))
        throw "makeGenCyl: profile curve must be flat on xy plane.";

    Surface surface;
    for (unsigned i = 0; i < sweep.size(); i++)
    {
        Matrix4f M = {{sweep[i].N, 0}, {sweep[i].B, 0}, {sweep[i].T, 0}, {sweep[i].V, 1}};
        for (const auto &p : profile)
            surface.VV.push_back((M * Vector4f(p.V, 1)).xyz()),
            surface.VN.push_back(M.getSubmatrix3x3(0,0).inverse().transposed() * -p.N);
    }
    addTriangles(surface.VF, sweep.size(), profile.size());
    return surface;
}
```

注意到$[N \quad B \quad T]$也是一个旋转矩阵，上面代码可以优化为

```
Surface makeGenCyl(const Curve &profile, const Curve &sweep)
{
    if (!checkFlat(profile))
        throw "makeGenCyl: profile curve must be flat on xy plane.";

    Surface surface;
    for (unsigned i = 0; i < sweep.size(); i++)
    {
        Matrix3f M = {sweep[i].N, sweep[i].B, sweep[i].T};
        for (const auto &p : profile)
            surface.VV.push_back(M * p.V + sweep[i].V),
            surface.VN.push_back(M * -p.N);
    }
    addTriangles(surface.VF, sweep.size(), profile.size());
    return surface;
}
```

绘制出的各种曲面如下。全部样例的图形见pictures文件夹。



附：变换向量和余向量

10.12 变换向量和余向量

我们已经明确：E^2 中的点$(x \quad y)$对应 3D 空间中的向量$[x \quad y \quad 1]^T$，向量$\begin{bmatrix} u \\ v \end{bmatrix}$对应 3D 空间中的向量$[u \quad v \quad 0]^T$。如果采用 3×3 的矩阵 $\boldsymbol{M}$(最后一行为$[0 \quad 0 \quad 1]$)进行 3D 空间变换：

$$T: \mathbf{R}^3 \to \mathbf{R}^3 : \boldsymbol{x} \mapsto \boldsymbol{MX} \tag{10-95}$$

那么 T 在 $W=1$ 平面上的投影在 E^2 中也有对应的映象，因此可以写成：

$$(T|E^2): E^2 \to E^2 : \boldsymbol{x} \mapsto \boldsymbol{Mx} \tag{10-96}$$

但是我们也注意到 T 可以作为一个变换**向量**，或者是 2D 欧几里得空间中的位移，它一般可以写为两个坐标，但通常用$[u \quad v \quad 0]^T$ 进行表示。因为这种向量的最后一个元素为 0，所以 $\boldsymbol{M}$ 的最后一列对于向量的变换没有影响。我们并不计算

$$\boldsymbol{M}\begin{bmatrix} u \\ v \\ 0 \end{bmatrix} \tag{10-97}$$

而是等价地计算

$$\begin{bmatrix} m_{1,1} & m_{1,2} & 0 \\ m_{2,1} & m_{2,2} & 0 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} u \\ v \\ 0 \end{bmatrix} \tag{10-98}$$

所得结果的第三个元素为 0。事实上，可以将这类向量当作 2 坐标的向量进行变换，只需要做简单计算

$$\begin{bmatrix} m_{1,1} & m_{1,2} \\ m_{2,1} & m_{2,2} \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} \tag{10-99}$$

因为这个原因，有时候我们说：对于由矩阵 $\boldsymbol{M}$ 相乘表示的欧式平面上的仿射变换，其向量的相关变换可以表示为

$$\overline{\boldsymbol{M}} = \begin{bmatrix} m_{1,1} & m_{1,2} \\ m_{2,1} & m_{2,2} \end{bmatrix} \tag{10-100}$$

对余向量如何计算？回想余向量的典型形式：

$$\phi_w : \mathbf{R}^2 \to \mathbf{R} : \boldsymbol{v} \mapsto \boldsymbol{w} \cdot \boldsymbol{v} \tag{10-101}$$

这里 $\boldsymbol{w}$ 是 $\mathbf{R}^2$ 中的某个向量。我们想采用同 T 一致的方法对 ϕ_w 进行变换。图 10-19 说明了为什么要这么做：我们经常构建某个形状的几何模型，并计算模型表面的法向量。假设 $\boldsymbol{n}$ 是一个表面法向量。对该几何模型实施“建模变换”T_M 将它放入 3D 空间，我们希望知道变换后的模型表面的法向量，以便计算光线 $\boldsymbol{v}$ 和表面法向的夹角，称变换后的表面法向量为 $\boldsymbol{m}$，现欲计算 $\boldsymbol{v} \cdot \boldsymbol{m}$。那么变换后的法向量 $\boldsymbol{m}$ 与原模型表面的法向量 $\boldsymbol{n}$ 之间有何对应关系呢？

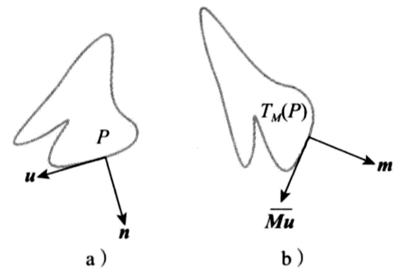


图 10-19　a）采用某种建模工具构建的几何形体；可计算得到点 P 的法向量 $\boldsymbol{n}$。向量 $\boldsymbol{u}$ 为点 P 处表面的切向；b）该形体被放入场景中，期间实施了平移、旋转和缩放。我们希望找到形体变换后 P 点的法向量 $\boldsymbol{m}$，且 $\boldsymbol{m}$ 与变换后的切向向量 $\overline{\boldsymbol{M}}\boldsymbol{u}$ 内积仍然为 0

根据表面法向的定义，原始模型上的法向量 $\boldsymbol{n}$ 与通过该点与模型表面相切的每一切向量 $\boldsymbol{u}$ 垂直。故新的法向量 $\boldsymbol{m}$ 也必须与所有变换后的切向量(与变换后模型表面相切)垂直。换句话说，对于物体表面上每个切向向量 $\boldsymbol{u}$，我们需要计算：

$$\boldsymbol{m} \cdot \overline{\boldsymbol{M}}\boldsymbol{u} = 0 \tag{10-102}$$

实际上可以更进一步，对于任意向量 $\boldsymbol{u}$，我们希望：

$$\boldsymbol{m} \cdot \overline{\boldsymbol{M}}\boldsymbol{u} = \boldsymbol{n} \cdot \boldsymbol{u} \tag{10-103}$$

这就是说，确保变换前某一向量和法向 $\boldsymbol{n}$ 之间的夹角同变换后该向量与 $\boldsymbol{m}$ 的夹角保持不变。

在求解之前，让我们先来看下面一些例子，对变换 T_1，与房子的底面垂直的向量(作为向量 $\boldsymbol{n}$)变换之后应仍与变换后的房子底面垂直。这可通过将其旋转 30°得到(见图 10-20)。

如果我们只是平移这栋房子，和其他向量一样，向量 $\boldsymbol{n}$ 并无变化。

但是当需要对房子进行错切变换时，如实施变换 T_3，情况如何呢？相应的向量变换仍然为错切变换，它使一个垂直向量变为倾斜。但对向量 $\boldsymbol{n}$ 而言，如果希望它仍然与房子底面保持垂直，就不必做任何改变(见图 10-21)。在这种情况下，我们看到，在是否需进行变换方面，余向量和向量存在不同之处。

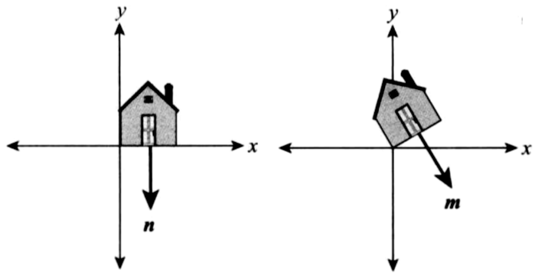


图 10-20　在形体旋转变换中，法向量和其他向量一样也发生旋转

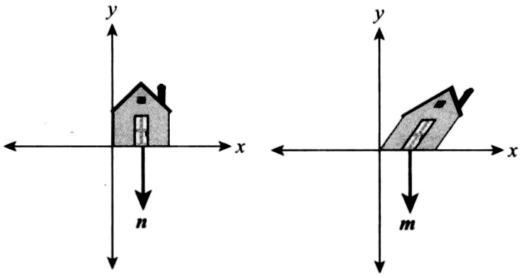


图 10-21　当房子的垂直面发生错切时，房子底面的法向保持不变

现在回到我们的问题：寻找一个向量 $\boldsymbol{m}$，对于每个可能的向量 $\boldsymbol{u}$，它满足：

$$\boldsymbol{m} \cdot (\overline{\boldsymbol{M}}\boldsymbol{u}) = \boldsymbol{n} \cdot \boldsymbol{u} \tag{10-104}$$

为了使推导更明显，交换向量的顺序，得到：

$$(\overline{\boldsymbol{M}}\boldsymbol{u}) \cdot \boldsymbol{m} = \boldsymbol{u} \cdot \boldsymbol{n} \tag{10-105}$$

由于 $\boldsymbol{a} \cdot \boldsymbol{b}$ 可以写作 $\boldsymbol{a}^{\mathrm{T}}\boldsymbol{b}$，上式可以改写为

$$(\overline{\boldsymbol{M}}\boldsymbol{u})^{\mathrm{T}}\boldsymbol{m} = \boldsymbol{u}^{\mathrm{T}}n \tag{10-106}$$

而$(\boldsymbol{AB})^{\mathrm{T}} = \boldsymbol{B}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}$，因此：

$$(\overline{\boldsymbol{M}}\boldsymbol{u})^{\mathrm{T}}\boldsymbol{m} = \boldsymbol{u}^{\mathrm{T}}\boldsymbol{n} \tag{10-107}$$

$$(\boldsymbol{u}^{\mathrm{T}}\ \overline{\boldsymbol{M}}^{\mathrm{T}})\boldsymbol{m} = \boldsymbol{u}^{\mathrm{T}}\boldsymbol{n} \tag{10-108}$$

$$\boldsymbol{u}^{\mathrm{T}}(\overline{\boldsymbol{M}}^{\mathrm{T}}\boldsymbol{m}) = \boldsymbol{u}^{\mathrm{T}}\boldsymbol{n} \tag{10-109}$$

最后一步基于矩阵相乘的结合性质。最后一个等式相当于：对于所有的向量 $\boldsymbol{u}$，$\boldsymbol{u} \cdot \boldsymbol{a} = \boldsymbol{u} \cdot \boldsymbol{b}$，此式当且仅当 $\boldsymbol{a} = \boldsymbol{b}$ 的时候才成立，即

$$\overline{\boldsymbol{M}}^{\mathrm{T}}\boldsymbol{m} = \boldsymbol{n} \tag{10-110}$$

所以

$$\boldsymbol{m} = (\overline{\boldsymbol{M}}^{\mathrm{T}})^{-1}\boldsymbol{n} \tag{10-111}$$

这里我们假设 $\overline{\boldsymbol{M}}$ 是可逆的。

因此我们可以得出：余向量 $\phi_{\boldsymbol{n}}$ 被变换为 $\phi_{(\overline{\boldsymbol{M}}^{\mathrm{T}})^{-1}\boldsymbol{n}}$。因为这个原因，逆转置常常叫作**余向量变换**或**法向变换**(因其常用于法向量变换)。注意如果我们将余向量写作行向量，则无需进行转置，但是需要将行向量*右乘* $\overline{\boldsymbol{M}}^{-1}$。

在通常数学表述中，法向变换沿相反的方向：取 $T_{\boldsymbol{M}}$ 陪域中一个法向量并生成定义域中的一个向量；该**伴随变换**的矩阵为 $\boldsymbol{M}^{\mathrm{T}}$。因为我们需要反向求解，所以取该矩阵的逆。